

DESCENSO COORDENADO. GRADIENTE PROYECTADO SIMPLE.

ALAN REYES-FIGUEROA
MÉTODOS NUMÉRICOS II

(AULA 29) 24.OCTUBRE.2024

Descenso Coordinado

Algoritmo: (Descenso Coordinado)

Inputs: $f : \mathbb{R}^n \rightarrow \mathbb{R}$ función de clase C^1 , con gradiente $\nabla f^2 f$; $\mathbf{x}_0 \in \mathbb{R}^n$.

Outputs: $\mathbf{x}$ punto crítico de f .

Obtain n , the dimension of the domain.

For $k = 0, 1, 2, \dots$ hasta que se cumpla un criterio de paro:

Define $j = k \pmod n$ the coordinate to step $j = k \% n$.

Set $\mathbf{d}_k = \mathbf{e}_j$.

Compute $-\nabla f(\mathbf{x}_k)[j] = -\frac{\partial f}{\partial x_j}(\mathbf{x}_k)$.

Set α_k by using any of Backtracking, Hessian or Cauchy strategies.

Define

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - (\alpha_k \nabla f(\mathbf{x}_k)[j]) \mathbf{e}_j \\ &= \mathbf{x}_k - (0, 0, \dots, 0, \alpha_k \frac{\partial f}{\partial x_j}(\mathbf{x}_k), 0, \dots, 0).\end{aligned}$$

Return $\mathbf{x}_{k+1}$.

Descenso Coordinado por Bloques

En ocasiones es conveniente dividir todas las n variables del dominio a optimizar en un conjunto de bloques

$$B_1, B_2, \dots, B_r.$$

El **descenso coordinado por bloques** imita la idea del descenso coordinado, sólo que en lugar de moverse a lo largo de una dirección $\mathbf{e}_j$ en cada iteración, este nuevo método se mueve a lo largo de todas las direcciones del bloque B_j .

Por ejemplo, si el bloque B_j consiste de las variables $\mathbf{x}_1, \mathbf{x}_7$ y $\mathbf{x}_{13}$, entonces la actualización

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) B_j$$

representa el ciclo

```
Define temp =  $\mathbf{x}_k$ .  
For  $i$  in  $B_j$ : (for  $i$  in  $[1, 7, 13]$ :)  
    temp = temp -  $(\alpha_k \nabla f(\mathbf{x}_k)[i]) \mathbf{e}_i$ ,  
Set  $\mathbf{x}_{k+1}$  = temp.
```

Descenso Coordinado por Bloques

Algoritmo: (*Descenso Coordinado por Bloques*)

Inputs: $f : \mathbb{R}^n \rightarrow \mathbb{R}$ función de clase C^1 , con gradiente $\nabla f^2 f$; $\mathbf{x}_0 \in \mathbb{R}^n$; una lista de bloques $[B_1, B_2, \dots, B_r]$ conformando una partición de las variables dominio.

Outputs: $\mathbf{x}$ punto crítico de f .

Obtain r , the number of blocks.

For $k = 0, 1, 2, \dots$ hasta que se cumpla un criterio de paro:

 Define $j = k \pmod{r}$ the coordinate Block to step $j = k \% r$.

 Set $\mathbf{d}_k = B_j$.

 Set α_k by using any of Backtracking, Hessian or Cauchy strategies.

 Define **temp** = $\mathbf{x}_k$.

 For i in B_j :

temp = **temp** - $(\alpha_k \nabla f(\mathbf{x}_k)[i]) \mathbf{e}_i$,

 Set $\mathbf{x}_{k+1}$ = **temp**.

Return $\mathbf{x}_{k+1}$.

Gradiente Proyectado

Otro método sencillo, pero útil, se obtiene cuando queremos restringir los valores de nuestra solución de optimización a un subdominio rectangular de $\mathbb{R}^n$.

En este caso, supongamos que deseamos nuestra solución objetivo $\mathbf{x}$ en el rectángulo

$$[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n] \subseteq \mathbb{R}^n.$$

En otras palabras, queremos

$$a_i \leq x_i \leq b_i, \text{ para toda } i = 1, 2, \dots, n. \quad (1)$$

El método de gradiente proyectado (simple) consiste en calcular la iteración del descenso gradiente usual

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k,$$

y luego obligar a que el nuevo punto $\mathbf{x}_{k+1}$ cumpla con todas las restricciones (1).

Gradiente Proyectado

En Python esto se logra fácilmente definiendo arrays

```
lo  = np.array([a1, a2, ..., an]),  
up  = np.array([b1, b2, ..., bn]),
```

y haciendo

```
xk+1[xk+1 < lo] = lo[xk+1 < lo],  
xk+1[xk+1 > up] = up[xk+1 > up].
```

Gradiente Projectado

Algoritmo: (*Gradiente Projectado*)

Inputs: $f : \mathbb{R}^n \rightarrow \mathbb{R}$ función de clase C^1 , con gradiente ∇f ; $\mathbf{x}_0 \in \mathbb{R}^n$; $\mathbf{lo} = [a_1, \dots, a_n]$ y $\mathbf{up} = [b_1, \dots, b_n]$ los límites del dominio rectangular.

Outputs: $\mathbf{x}$ punto crítico de f .

For $k = 0, 1, 2, \dots$ hasta que se cumpla un criterio de paro:

Find $\mathbf{d}_k$ a descent direction.

Compute α_k by using any of Backtracking or other method.

Define

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k,$$

Apply

$$\mathbf{x}_{k+1}[\mathbf{x}_{k+1} < \mathbf{lo}] = \mathbf{lo}[\mathbf{x}_{k+1} < \mathbf{lo}],$$

$$\mathbf{x}_{k+1}[\mathbf{x}_{k+1} > \mathbf{up}] = \mathbf{up}[\mathbf{x}_{k+1} > \mathbf{up}].$$

Return $\mathbf{x}_{k+1}$.